Models of natural computation : gene assembly and membrane systems
Brijder, R.

Citation
Brijder, R. (2008, December 3). *Models of natural computation : gene assembly and membrane systems*. *IPA Dissertation Series*. Retrieved from https://hdl.handle.net/1887/13345

Version:	Corrected Publisher's Version
License:	Licence agreement concerning inclusion of doctoral thesis in the Institutional Repository of the University of Leiden
Downloaded from:	https://hdl.handle.net/1887/13345

Note: To cite this publication please use the final published version (if applicable).

Chapter 3

Strategies of Loop Recombination in Ciliates

Abstract
The concept of breakpoint graph, known from the theory of sorting by reversal, has been successfully applied in the theory of gene assembly in ciliates. We further investigate its usage for gene assembly, and show that the graph allows for an efficient characterization of the possible orders of loop recombination operations (one of the three types of molecular operations that accomplish gene assembly) for a given gene during gene assembly. The characterization is based on spanning trees within a graph built upon the connected components in the breakpoint graph. We work in the abstract and more general setting of so-called legal strings.

3.1 Introduction

Gene assembly is an involved DNA transformation process in ciliates (a large group of single cell organisms) which transforms a nucleus (the micronucleus) into a functionally different nucleus (the macronucleus). The process is accomplished using three types of DNA splicing operations, which operate on special DNA sequences called pointers. Each pointer can be seen as a breakpoint with a 'tag' which specifies how the splicing should be done, ensuring that the end result is fixed. The process however is not deterministic: for every gene in its micronuclear form, there can be several sequences of operations, called strategies, to transform this gene to its macronuclear form. For a given micronuclear gene, strategies may differ in the number of operations. It has been shown however that the number of loop recombination operations is independent of the chosen strategy [14, 12], and that this number can be efficiently calculated [6, 5].

In this chapter we characterize for a given set of pointers D, whether or not there is a strategy that applies loop recombination (called string negative rule in the formal model that we use) on exactly these pointers. This result depends

heavily on the reduction graph, which is motivated by the breakpoint graph in the theory of sorting by reversal [21, 23, 1] since it adopts the concept of reality-and-desire for DNA sequences with breakpoints. More specifically, we define a graph, called the pointer-component graph, 'on top of' the reduction graph, thereby depicting the distribution of pointers over the connected components of the reduction graph [6, 5]. We show that one can apply loop recombination on the pointers in D exactly when D forms a spanning tree in the pointer-component graph. This characterization implies an efficient algorithm. Also, we characterize in which order the pointers of D can possibly be applied in strategies.

This chapter is organized as follows. In Section 3.2 we recall basic notions and terminology mainly concerning strings and graphs, and in Section 3.3 we recall a formal model of the gene assembly process: the string pointer reduction system. In Section 3.4 we recall the notion of reduction graph and some theorems related to this notion. In Section 3.5 we define the pointer-component graph, a graph that depends on the reduction graph, and we discuss an operation on this graph that captures the application of string pointer rules. In Section 3.6 we show that the spanning trees of the pointer-component graphs reveal crucial information concerning applicability of string negative rules. Section 3.7 shows that merging and splitting of vertices in pointer-component graphs relate to the removal of pointers. Using the results of Sections 3.6 and 3.7, we characterize in Section 3.8 for a given set of pointers D, whether or not there is a strategy that applies string negative rules on exactly these pointers. Section 3.9 strengthens results of Section 3.8 by also characterizing in which order the string negative rules can be applied on the pointers. We conclude this chapter with Section 3.10. A conference edition of this chapter, containing selected results without proofs, was presented at CompLife '06 [3].

3.2 Basic Notions and Notation

In this section we recall some basic notions concerning functions, strings, and graphs. We do this mainly to fix the basic notation and terminology.

The restriction of f to a subset A of X is denoted by $f|A$, and for $D \subseteq X$ we denote by $f(D)$ the set $\{f(x) \mid x \in D\}$. Let $\prec$ be a binary relation over a finite set X. Then x_i and x_j in X are called *independent*, if they are incomparable in the transitive closure of $\prec$. A *topological ordering* of $\prec$ is a linear ordering $(x_1, \ldots, x_n)$ of X such that if $x_i \prec x_j$, then $i < j$.

We will use λ to denote the empty string. For strings u and v, we say that v is a *substring of* u if $u = w_1 v w_2$, for some strings w_1, w_2; we also say that v *occurs in* u.

Let Σ and Δ be alphabets. For $\Gamma \subseteq \Sigma$, we denote by $\mathrm{erase}_\Gamma : \Sigma^* \to \Delta^*$ the homomorphism defined by

$$\varphi(a) = \begin{cases} a & \text{if } a \notin \Gamma \\ \lambda & \text{if } a \in \Gamma \end{cases},$$

for all $a \in \Sigma$.

We now turn to graphs. We will only consider undirected graphs. A *graph* is a tuple $G = (V, E)$, where V is a finite set and $E \subseteq \{\{x, y\} \mid x, y \in V\}$. The elements of V are called *vertices* and the elements of E are called *edges*. We also write $o(G) = |V|$. We allow $x = y$, and therefore edges can be of the form $\{x, x\} = \{x\}$ — an edge of this form should be seen as an edge connecting x to x, i.e., a 'loop' for x. Vertex x is *isolated* in G if there is no edge e of G with $x \in e$. The *restriction of G to $E' \subseteq E$*, denoted by $G|_{E'}$, is (V, E').

A *multigraph* is a graph $G = (V, E, \epsilon)$, where parallel edges are possible. Therefore, E is a finite set of edges and $\epsilon : E \to \{\{x, y\} \mid x, y \in V\}$ is the *endpoint mapping*. Clearly, if ϵ is injective, then such a multigraph is equivalent to a graph. We let MGr denote the set of all multigraphs.

A *2-edge coloured graph* is a graph $G = (V, E_1, E_2, f, s, t)$ where E_1 and E_2 are two finite (not necessarily disjoint) sets of edges, $s, t \in V$ are two distinct vertices called the *source vertex* and the *target vertex*, respectively, and there is a vertex labelling function $f : V \backslash \{s, t\} \to \Gamma$ for some finite set Γ. The elements of Γ are the *vertex labels*. We use $\mathsf{2EGr}$ to denote the set of all 2-edge coloured graphs.

Notions such as isomorphisms, paths, connectedness, and trees for graphs carry over to these two types of graphs. For example, for a multigraph $G = (V, E, \epsilon)$ and $E' \subseteq E$, we have $G|_{E'} = (V, E', \epsilon|E')$. Care must be taken for isomorphisms. Multigraphs $G = (V, E, \epsilon)$ and $G' = (V', E, \epsilon')$ are *isomorphic* if there is a bijection $\alpha : V \to V'$ such that $\alpha\epsilon = \epsilon'$, or more precisely, for $e \in E$, $\epsilon(e) = \{v_1, v_2\}$ implies $\epsilon'(e) = \{\alpha(v_1), \alpha(v_2)\}$. Note that the sets of edges of G and G' are identical. Also, 2-edge coloured graphs $G = (V, E_1, E_2, f, s, t)$ and $G' = (V', E'_1, E'_2, f', s', t')$ are *isomorphic* if there is a bijection $\alpha : V \to V'$ such that $\alpha(s) = s'$, $\alpha(t) = t'$, $f(v) = f'(\alpha(v))$ for all $v \in V$, and $\{x, y\} \in E_i$ iff $\{\alpha(x), \alpha(y)\} \in E'_i$, for all $x, y \in V$, and $i \in \{1, 2\}$.

For 2-edge coloured graphs G, we say that a path $\pi = e_1 e_2 \cdots e_n$ in G is an *alternating path in G* if, for $1 \leq i < n$, both $e_i \in E_1$ and $e_{i+1} \in E_2$, or the other way around.

3.3 String Pointer Reduction System

Three (almost) equivalent formal models for gene assembly were considered in [15, 11, 12]. In this section we briefly recall the one that we will use in this chapter: the string pointer reduction system. For a detailed motivation and other results concerning this model we refer to [12].

We fix $\kappa \geq 2$, and define the alphabet $\Delta = \{2, 3, \ldots, \kappa\}$. For $D \subseteq \Delta$, we define $\bar{D} = \{\bar{a} \mid a \in D\}$ with $D \cap \bar{D} = \varnothing$, and we define $\Pi = \Delta \cup \bar{\Delta}$. The elements of Π will be called *pointers*. Since we work in the general framework of legal strings, the exact identities of the elements in Δ is irrelevant, in fact, any finite set Δ would suffice. However, we respect the convention of denoting a pointer by an integer larger than 1 with possibly a bar. The name 'pointer' is lent from computer science due to its similarities with pointers defined here (see, e.g., Chapter 16 in [12]).

$5 \quad 4 \quad 3 \quad 7 \quad 2 \quad 5 \quad 6 \quad 2 \quad \bar{7} \quad 3 \quad 4 \quad 6$

Figure 3.1: Sequence of pointers represented by the legal string $u = 54372562\bar{7}346$.

We use the 'bar operator' to move from Δ to $\bar{\Delta}$ and back from $\bar{\Delta}$ to Δ. Hence, for $p \in \Pi$, $\bar{\bar{p}} = p$. For a string $u = x_1 x_2 \cdots x_n$ with $x_i \in \Pi$, the *inverse* of u is the string $\bar{u} = \bar{x}_n \bar{x}_{n-1} \cdots \bar{x}_1$. For $p \in \Pi$, we define $\mathbf{p}$ to be p if $p \in \Delta$, and $\bar{p}$ if $p \in \bar{\Delta}$, i.e., $\mathbf{p}$ is the 'unbarred' variant of p. The *domain* of a string $v \in \Pi^*$ is $\mathrm{dom}(v) = \{\mathbf{p} \mid p \text{ occurs in } v\}$. A *legal string* is a string $u \in \Pi^*$ such that for each $p \in \Pi$ that occurs in u, u contains exactly two occurrences from $\{p, \bar{p}\}$. For a pointer p and a legal string u, if both p and $\bar{p}$ occur in u then we say that both p and $\bar{p}$ are *positive* in u; if on the other hand only p or only $\bar{p}$ occurs in u, then both p and $\bar{p}$ are *negative* in u.

Let $u = x_1 x_2 \cdots x_n$ be a legal string with $x_i \in \Pi$ for $1 \le i \le n$. For a pointer $p \in \Pi$ such that $\{x_i, x_j\} \subseteq \{p, \bar{p}\}$ and $1 \le i < j \le n$, the *p-interval of u* is the substring $x_i x_{i+1} \cdots x_j$. Two distinct pointers $p, q \in \Pi$ *overlap* in u if both $\mathbf{q} \in \mathrm{dom}(I_p)$ and $\mathbf{p} \in \mathrm{dom}(I_q)$, where I_p (I_q, resp.) is the p-interval (q-interval, resp.) of u.

Example 1
String $u = \bar{4}37\bar{7}\bar{4}3$ is a legal string. However, $v = 424$ is not a legal string. Also, $\mathrm{dom}(u) = \{3, 4, 7\}$ and $\bar{u} = \bar{3}47\bar{7}\bar{3}4$. The 3-interval of u is $37\bar{7}\bar{4}3$, and pointers 3 and 4 overlap in u.

A legal string is a representation of a sequence of pointers. The legal string $u = 54372562\bar{7}346$ corresponds to the sequence of pointers in Figure 3.1. Each gene in the micronucleus in ciliates can be represented by such a legal string. For example, the legal string $344567567 89\bar{3}\bar{2}289$ corresponds to the micronuclear form of the gene that corresponds to the actin protein in the stichotrich *Sterkiella nova* (see [22, 12, 8]). Gene assembly transforms each gene in micronuclear form to its macronuclear form by three splicing operations which operate on the pointers. These three operations are formally defined on legal strings through the string pointer reduction system, where each is defined on a specific pattern of the pointers.

The string pointer reduction system consists of three types of reduction rules operating on legal strings. For all $p, q \in \Pi$ with $\mathbf{p} \neq \mathbf{q}$, we define:

- the *string negative rule* for p by $\mathbf{snr}_p(u_1 p p u_2) = u_1 u_2$,
- the *string positive rule* for p by $\mathbf{spr}_p(u_1 p u_2 \bar{p} u_3) = u_1 \bar{u}_2 u_3$,
- the *string double rule* for p, q by $\mathbf{sdr}_{p,q}(u_1 p u_2 q u_3 p u_4 q u_5) = u_1 u_4 u_3 u_2 u_5$,

where $u_1, u_2, \ldots, u_5$ are arbitrary (possibly empty) strings over Π. We also define $Snr = \{\mathbf{snr}_p \mid p \in \Pi\}$, $Spr = \{\mathbf{spr}_p \mid p \in \Pi\}$ and $Sdr = \{\mathbf{sdr}_{p,q} \mid p, q \in \Pi, \mathbf{p} \neq \mathbf{q}\}$ to be the sets containing all the reduction rules of a specific type.

Note that each of these rules is defined only on legal strings that satisfy the given form. For example, $\mathbf{spr}_{\bar{2}}$ is defined on the legal string $\bar{2}323$, however $\mathbf{spr}_2$ is not defined on this legal string. Also note that for every non-empty legal string there is at least one reduction rule applicable. Indeed, every non-empty legal string for which no string positive rule and no string double rule is applicable must have only non-overlapping negative pointers, thus there is a string negative rule which is applicable. This is formalized in Theorem 1 below.

The *domain* of a reduction rule ρ, denoted by $\mathrm{dom}(\rho)$, is defined by $\mathrm{dom}(\mathbf{snr}_p) = \mathrm{dom}(\mathbf{spr}_p) = \{\mathbf{p}\}$ and $\mathrm{dom}(\mathbf{sdr}_{p,q}) = \{\mathbf{p}, \mathbf{q}\}$ for $p, q \in \Pi$. For a composition $\varphi = \varphi_n \ \cdots \ \varphi_2 \ \varphi_1$ of reduction rules $\varphi_1, \varphi_2, \ldots, \varphi_n$, the *domain*, denoted by $\mathrm{dom}(\varphi)$, is defined by $\mathrm{dom}(\varphi) = \mathrm{dom}(\varphi_1) \cup \mathrm{dom}(\varphi_2) \cup \cdots \cup \mathrm{dom}(\varphi_n)$.

Example 2
The domain of $\varphi = \mathbf{snr}_2 \ \mathbf{spr}_{\bar{4}} \ \mathbf{sdr}_{7,5} \ \mathbf{snr}_{\bar{9}}$ is $\mathrm{dom}(\varphi) = \{2, 4, 5, 7, 9\}$.

Let $S \subseteq \{Snr, Spr, Sdr\}$. Then a composition φ of reduction rules from S is called an *(S-)reduction.* Let u be a legal string. We say that φ is a *reduction of* u, if φ is a reduction and φ is applicable to (i.e., defined on) u. A *successful reduction* φ *of* u is a reduction of u such that $\varphi(u) = \lambda$. We then also say that φ is *successful for* u. We say that u is *successful in* S if there is a successful S-reduction of u. Note that if φ is a reduction of u, then $\mathrm{dom}(\varphi) = \mathrm{dom}(u) \backslash \mathrm{dom}(\varphi(u))$.

Example 3
Again let $u = \bar{4}37\bar{7}\bar{4}3$. Then $\varphi_1 = \mathbf{sdr}_{\bar{4},3} \ \mathbf{spr}_7$ is a successful $\{Spr, Sdr\}$-reduction of u. However, both $\varphi_2 = \mathbf{snr}_3 \ \mathbf{spr}_7$ and $\varphi_3 = \mathbf{snr}_8$ are *not* reductions of u.

Example 4
If we again consider the legal string $3445675678 9\bar{3}\bar{2}289$, which represents the micronuclear form of the gene corresponding to the actin protein in the stichotrich *Sterkiella nova*, then $\mathbf{spr}_3 \ \mathbf{sdr}_{8,9} \ \mathbf{snr}_7 \ \mathbf{sdr}_{5,6} \ \mathbf{snr}_4 \ \mathbf{spr}_{\bar{2}}$ is a successful reduction of this legal string. Therefore, this sequence of operations transforms the gene from its micronuclear form to its macronuclear form.

We say that a linear ordering $L = (p_1, \ldots, p_n)$ of a subset of $\mathrm{dom}(\varphi)$ is the *Snr-order of* φ, if $\varphi = \varphi_{n+1} \ \mathbf{snr}_{\tilde{p}_n} \ \varphi_n \ \mathbf{snr}_{\tilde{p}_{n-1}} \ \cdots \ \varphi_2 \ \mathbf{snr}_{\tilde{p}_1} \ \varphi_1$ for some (possible empty) $\{Spr, Sdr\}$-reductions $\varphi_1, \varphi_2, \ldots, \varphi_{n+1}$ and $\tilde{p_i} \in \{p_i, \bar{p_i}\}$ for $1 \leq i \leq n$ and $n \geq 0$. Moreover, we define $\mathrm{snrdom}(\varphi) = \{p_1, \ldots, p_n\}$.

Example 5
The *Snr*-order of $\varphi = \mathbf{snr}_2 \ \mathbf{spr}_{\bar{4}} \ \mathbf{sdr}_{7,5} \ \mathbf{snr}_{\bar{9}}$ is $(9, 2)$, and $\mathrm{snrdom}(\varphi) = \{2, 9\}$.

Since for every (non-empty) legal string there is an applicable reduction rule, by iterating this argument, we have the following well-known result.

Theorem 1
For every legal string u there is a successful reduction of u.

3.4 Reduction Graph

In this section we recall the definition of reduction graph and some results concerning this graph. First we give the definition of pointer removal operations on strings, see also [6].

Definition 2
For a subset $D \subseteq \Delta$, the D-removal operation, denoted by rem_D, is defined by $\mathrm{rem}_D = \mathrm{erase}_{D \cup \bar{D}}$. We also refer to rem_D operations, for all $D \subseteq \Delta$, as *pointer removal operations.* ■

Note that for each legal string u, $\mathrm{rem}_D(u)$ is a legal string.

Example 6
Let $u = 5437256 2\bar{7}346$ be a legal string. We will use this legal string as our running example for this chapter. For $D = \{4, 6, 7, 9\}$, we have $\mathrm{rem}_D(u) = 532523$.

The next lemma is an easy consequence of Lemma 8 from [6]. It cannot be extended to *Snr* rules: if $\mathbf{snr}_p$ is applicable to $\mathrm{rem}_D(u)$, then it is not necessarily applicable to u.

Lemma 3
Let u be a legal string, let φ be a composition of reduction rules that does not contain string negative rules, and let $D = \mathrm{dom}(u) \backslash \mathrm{dom}(\varphi)$. Then φ is a reduction of u iff φ is a (successful) reduction of $\mathrm{rem}_D(u)$.

The string negative rules in a reduction can be 'postponed' without affecting the applicability. More precisely, if $\varphi = \varphi_2\ \rho\ \mathbf{snr}_p\ \varphi_1$ is a reduction of a legal string u, with $p \in \Pi$, ρ a string positive rule or string double rule, and φ_1, φ_2 arbitrary compositions of reduction rules, then there is a $\tilde{p} \in \{p, \bar{p}\}$ such that $\varphi_2\ \mathbf{snr}_{\tilde{p}}\ \rho\ \varphi_1$ is a reduction of u. Thus we can separate each reduction into a sequence without *Snr* rules, and a tail of *Snr* rules. We often use this 'normal form'.

Example 7
We continue the example. Since $\varphi = \mathbf{snr}_6\ \mathbf{snr}_2\ \mathbf{spr}_{\bar{7}}\ \mathbf{snr}_4\ \mathbf{sdr}_{5,3}$ is a successful reduction of u, it follows that $\varphi' = \mathbf{snr}_6\ \mathbf{snr}_2\ \mathbf{snr}_{\tilde{4}}\ \mathbf{spr}_{\bar{7}}\ \mathbf{sdr}_{5,3}$ is also a successful reduction of u for some $\tilde{4} \in \{4, \bar{4}\}$. One can verify that we can take $\tilde{4} = 4$. However, $\varphi'' = \mathbf{snr}_4\ \mathbf{snr}_6\ \mathbf{snr}_2\ \mathbf{spr}_{\bar{7}}\ \mathbf{sdr}_{5,3}$ is *not* a successful reduction of u. If we consider the legal string $7\bar{2}\bar{2}\bar{7}$, for which $\mathbf{spr}_7\ \mathbf{snr}_{\bar{2}}$ is a successful reduction, then by postponing the string negative rule, $\mathbf{snr}_2\ \mathbf{spr}_7$ is also a successful reduction of this legal string.

Figure 3.2 illustrates Lemma 3 when φ is in this normal form: $\varphi = \varphi_2\ \varphi_1$ is a successful reduction of u, where φ_1 is a $\{Spr, Sdr\}$-reduction and φ_2 is a $\{Snr\}$-reduction with $\mathrm{dom}(\varphi_2) = D$.

We are now ready to recall the definition of reduction graph. It was introduced in [6], and we restate it here in a less general form. A reduction graph is a 2-edge

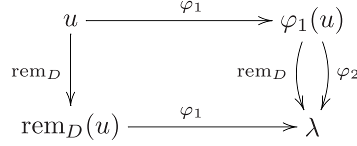

Figure 3.2: An illustration of Lemma 3: $\varphi = \varphi_2\ \varphi_1$ is a successful reduction of u, where φ_1 is a $\{Spr, Sdr\}$-reduction and φ_2 is a $\{Snr\}$-reduction with $\mathrm{dom}(\varphi_2) = D$.

coloured graph where the two types of edges are called *reality edges* and *desire edges*. Moreover, all vertices, except for two distinct vertices s and t, are labelled by an element from Δ. Recall that the physical representation of our running example $u = 54372562\bar{7}346$ is given in Figure 3.1. The reduction graph is defined in such a way that (1) each (occurrence of a) pointer of u appears twice (in unbarred form) as a vertex in the graph to represent both sides of the pointer in Figure 3.1, (2) the reality edges (depicted as 'double edges' to distinguish them from the desire edges) represent the segments between the pointers, (3) the desire edges represent which segments should be glued to each other when operations are applied on the corresponding pointers. Positive pointers are connected by crossing desire edges (cf. pointer 7 in Figure 3.3), while negative pointers are connected by parallel desire edges. We refer to [6] for a more elaborate motivation and for more examples and results concerning this graph. The notion is similar to the breakpoint graph (or reality-and-desire diagram) known from another branch of DNA processing theory called sorting by reversal, see e.g. [23] and [21].

Definition 4
Let $u = p_1 p_2 \cdots p_n$ with $p_1, \ldots, p_n \in \Pi$ be a legal string. The *reduction graph of* u, denoted by $\mathcal{R}_u$, is a 2-edge coloured graph (V, E_1, E_2, f, s, t), where

$$V = \{I_1, I_2, \ldots, I_n\}\ \cup\ \{I'_1, I'_2, \ldots, I'_n\}\ \cup\ \{s, t\},$$

$$E_1 = \{e_0, e_1, \ldots, e_n\} \text{ with } e_i = \{I'_i, I_{i+1}\} \text{ for } 1 < i < n, e_0 = \{s, I_1\}, e_n = \{I'_n, t\},$$

$$\begin{aligned} E_2 = \quad & \{\{I'_i, I_j\}, \{I_i, I'_j\} \mid i, j \in \{1, 2, \ldots, n\} \text{ with } i \neq j \text{ and } p_i = p_j\}\ \cup \\ & \{\{I_i, I_j\}, \{I'_i, I'_j\} \mid i, j \in \{1, 2, \ldots, n\} \text{ and } p_i = \bar{p}_j\}, \text{ and} \end{aligned}$$

$$f(I_i) = f(I'_i) = \mathbf{p}_i \text{ for } 1 \leq i \leq n.$$

■

The edges of E_1 are called *reality edges*, and the edges of E_2 are called *desire edges*. Notice that for each $p \in \mathrm{dom}(u)$, the reduction graph of u has exactly two desire edges containing vertices labelled by p.

In depictions of reduction graphs, we will represent the vertices (except for s and t) by their labels, because the exact identities of the vertices are not essential

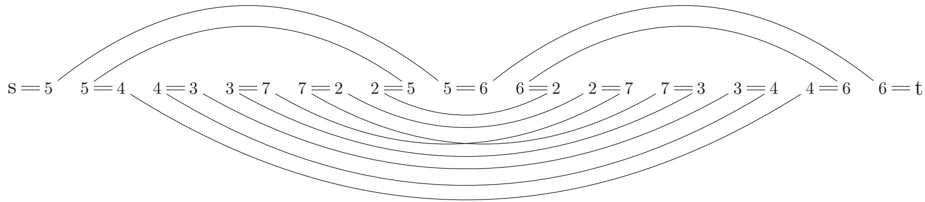

Figure 3.3: The reduction graph $\mathcal{R}_u$ of u from Example 8.

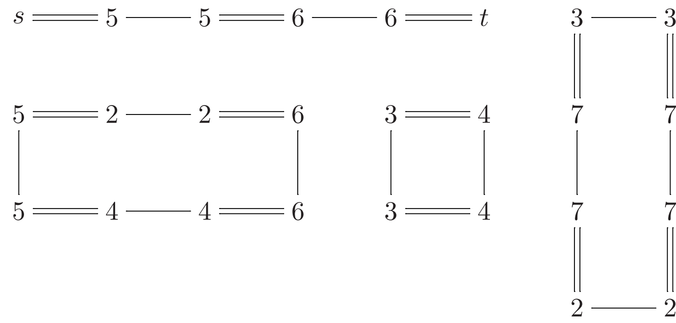

Figure 3.4: The reduction graph of Figure 3.3.

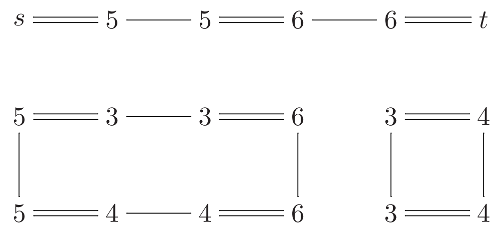

Figure 3.5: The reduction graph $\mathcal{R}_{rem_{\{2,7\}}(u)}$ from the Example.

for the problems considered in this chapter. We will also depict reality edges as 'double edges' to distinguish them from the desire edges.

Example 8
We continue the example. Reduction graph $\mathcal{R}_u$ is given in Figure 3.3. The same graph is again depicted in Figure 3.4 – we have only rearranged the vertices. Also, $\mathcal{R}_{rem_{\{2,7\}}(u)}$ is given in Figure 3.5.

Each reduction graph has a connected component with a linear structure containing both the source and the target vertex [6]. This connected component is called the *linear component* of the reduction graph. The other connected components are called *cyclic components* because of their structure.

The definition of reduction functions and the remaining results are also taken from [6]. The p-reduction function removes vertices labelled by p and 'contracts' alternating paths via these vertices into a single edge.

Definition 5
For each vertex label p, we define the *p-reduction function* $rf_p : \mathsf{2EGr} \to \mathsf{2EGr}$, for $G = (V, E_1, E_2, f, s, t) \in \mathsf{2EGr}$, by

$$rf_p(G) = (V', (E_1 \backslash E_{rem}) \cup E_{add}, E_2 \backslash E_{rem}, f|V', s, t),$$

with

$$\begin{aligned}
V' &= \{s,t\} \cup \{v \in V \backslash \{s,t\} \mid f(v) \neq p\}, \\
E_{rem} &= \{e \in E_1 \cup E_2 \mid f(x) = p \text{ for some } x \in e\}, \text{and} \\
E_{add} &= \{\{y_1, y_2\} \mid e_1 e_2 \cdots e_n \text{ with } n > 2 \text{ is an alternating path in } G \\
&\qquad \text{with } y_1 \in e_1,\ y_2 \in e_n,\ f(y_1) \neq p \neq f(y_2), \text{ and} \\
&\qquad f(x) = p \text{ for all } x \in e_i,\ 1 < i < n\}.
\end{aligned}$$

■

Reduction functions commute under composition. Thus, for a reduction graph $\mathcal{R}_{rem_D(u)}$ and pointers p and q, we have

$$(rf_q\ rf_p)(\mathcal{R}_u) = (rf_p\ rf_q)(\mathcal{R}_u).$$

Any reduction can be simulated, on the level of reduction graphs, by a sequence of reduction functions with the same domain, cf. [6, Theorem 17].

Theorem 6
Let u be a legal string, and let φ be a reduction of u. Then

$$(rf_{p_n}\ \cdots\ rf_{p_2}\ rf_{p_1})(\mathcal{R}_u) \approx \mathcal{R}_{\varphi(u)},$$

where $\mathrm{dom}(\varphi) = \{p_1, p_2, \ldots, p_n\}$.

The next lemma is an easy consequence from Lemma 22 and Lemma 23 in [6].

$$s \Longleftarrow \mathbf{p} \longrightarrow \mathbf{p} \Longleftarrow \mathbf{q} \longrightarrow \mathbf{q} \Longleftarrow \mathbf{p} \longrightarrow \mathbf{p} \Longleftarrow \mathbf{q} \longrightarrow \mathbf{q} \Longleftarrow t$$

Figure 3.6: The reduction graph of $pq\bar{p}q$ (and $pqpq$).

Lemma 7
Let u be a legal string and let $p \in \Pi$. Then $\mathcal{R}_u$ has a cyclic component C consisting of only vertices labelled by $\mathbf{p}$ iff either pp or $\bar{p}\bar{p}$ is a substring of u. Moreover, if C exists, then it has exactly two vertices.

One of the motivations for the reduction graph is the easy determination of the number of string negative rules needed in each successful reduction [6, Theorem 26].

Theorem 8
Let N be the number of cyclic components in the reduction graph of legal string u. Then every successful reduction of u has exactly N string negative rules.

Example 9
We continue the example. Since $\mathcal{R}_u$ has three cyclic components, by Theorem 8, every successful reduction φ of u has exactly three string negative rules. For example $\varphi = \mathbf{snr}_6\ \mathbf{snr}_2\ \mathbf{spr}_{\bar{7}}\ \mathbf{snr}_4\ \mathbf{sdr}_{5,3}$ is a successful reduction of u. Indeed, φ has exactly three string negative rules. Alternatively, $\mathbf{snr}_6\ \mathbf{snr}_4\ \mathbf{snr}_3\ \mathbf{spr}_2\ \mathbf{spr}_5\mathbf{spr}_7$ is also a successful reduction of u, with a different number of (**spr** and **sdr**) operations.

The previous theorem and example should clarify that the reduction graph reveals crucial properties concerning the string negative rule. We now further investigate the string negative rule, and show that many more properties of this rule can be revealed using the reduction graph.

However, the reduction graph does not seem to be well suited to prove properties of the string positive rule and string double rule. If we for example consider legal strings $u = pq\bar{p}q$ and $v = pqpq$ for some distinct $p, q \in \Pi$, then u has a unique successful reduction $\varphi_1 = \mathbf{spr}_{\bar{q}}\ \mathbf{spr}_p$ and v has a unique successful reduction $\varphi_2 = \mathbf{sdr}_{p,q}$. Thus u must necessarily be reduced by string positive rules, while v must necessarily be reduced by a string double rule. However, the reduction graph of u and the reduction graph of v are isomorphic, as shown in Figure 3.6. Also, whether or not pointers overlap is not preserved by reduction graphs. For example, the reduction graphs of legal strings $pqp\bar{r}qr$ and $pqr\bar{p}qr$ for distinct pointers p, q and r are isomorphic, however p and r do not overlap in the first legal string, but they do overlap in the latter legal string.

The next lemma is an easy consequence of Lemma 3 and Theorem 8.

Lemma 9
Let u be a legal string, and let $D \subseteq \mathrm{dom}(u)$. There is a $\{Spr, Sdr\}$-reduction φ of u with $\mathrm{dom}(\varphi(u)) = D$ iff $\mathcal{R}_{rem_D(u)}$ does not contain cyclic components.

Proof
There is a $\{Spr, Sdr\}$-reduction φ of u with $\mathrm{dom}(\varphi(u)) = D$ iff there is a successful $\{Spr, Sdr\}$-reduction of $\mathrm{rem}_D(u)$ (by Lemma 3) iff $\mathcal{R}_{rem_D(u)}$ does not contain cyclic components (by Theorem 8). ■

Using Theorem 8 and Lemma 9 we obtain a first characterization of the sets of pointers that are used in string negative rules in successful reductions.

Lemma 10
Let u be a legal string, and let $D \subseteq \mathrm{dom}(u)$. There is a successful reduction φ of u with $\mathrm{snrdom}(\varphi) = D$ iff $\mathcal{R}_{rem_D(u)}$ and $\mathcal{R}_u$ have 0 and $|D|$ cyclic components, respectively.

Proof
We first prove the forward implication. Since we can postpone the string negative rules, there is a successful reduction $\varphi' = \varphi'_2\varphi'_1$ of u, where φ'_1 is a $\{Spr, Sdr\}$-reduction and φ'_2 is a $\{Snr\}$-reduction with $\mathrm{dom}(\varphi'_2) = D$. By Lemma 9, $\mathcal{R}_{rem_D(u)}$ does not contain cyclic components. By Theorem 8, $\mathcal{R}_u$ has $|D|$ cyclic components.

We now prove the reverse implication. By Lemma 9, there is a successful reduction $\varphi = \varphi_2\ \varphi_1$ of u, where φ_1 is a $\{Spr, Sdr\}$-reduction and $\mathrm{dom}(\varphi_2) = D$. Since $\mathcal{R}_u$ has $|D|$ cyclic components, by Theorem 8, every pointer in D is used in a string negative rule, and thus φ_2 is a $\{Snr\}$-reduction. ■

3.5 Pointer-Component Graphs

If it is clear from the context which legal string u is meant, we will denote by ζ the set of connected components of the reduction graph of u. We now define a graph on ζ that we will use throughout the rest of this chapter. The graph represents how the labels of a reduction graph are distributed among its connected components. This graph is particularly useful in determining which sets D of pointers correspond to strategies that apply loop recombination operations on exactly the pointers of D.

Definition 11
Let u be a legal string. The *pointer-component graph of u (or of $\mathcal{R}_u$)*, denoted by $\mathcal{PC}_u$, is a multigraph (ζ, E, ϵ), where $E = \mathrm{dom}(u)$ and ϵ is, for $e \in E$, defined by $\epsilon(e) = \{C \in \zeta \mid C \text{ contains vertices labelled by } e\}$. ■

Note that for each $e \in \mathrm{dom}(u)$, there are exactly two desire edges connecting vertices labelled by e, thus $1 \leq |\epsilon(e)| \leq 2$, and therefore ϵ is well defined.

Example 10
We continue the example. Consider $\mathcal{R}_u$ shown in Figure 3.4. Let us define C_1 to be the cyclic component with a vertex labelled by 7, C_2 to be the cyclic component with a vertex labelled by 5, C_3 to be the third cyclic component, and R to be the linear component. Then $\zeta = \{C_1, C_2, C_3, R\}$. The pointer-component graph $\mathcal{PC}_u = (\zeta, \mathrm{dom}(u), \epsilon)$ of u is given in Figure 3.7. As C_1 contains all four vertices labelled by 7, this results in a loop for C_1 in $\mathcal{PC}_u$.

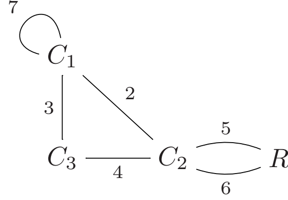

Figure 3.7: The graph $\mathcal{PC}_u$ from the Example.

By the definition of pointer-component graph and Theorem 8, every successful reduction of a legal string u has exactly $o(\mathcal{PC}_u) - 1$ string negative rules (recall that $o(\mathcal{PC}_u)$ denotes the number of vertices of $\mathcal{PC}_u$). Let ρ be a reduction rule applicable to u. Then by Theorem 1, there is a successful reduction φ' of $\rho(u)$. Hence, $\varphi'\rho$ is a successful reduction of u. Thus, if ρ is a string positive rule or string double rule, then $o(\mathcal{PC}_{\rho(u)}) = o(\mathcal{PC}_u)$, and if ρ is a string negative rule, then $o(\mathcal{PC}_{\rho(u)}) = o(\mathcal{PC}_u) - 1$. Thus we have the following result.

Theorem 12
Let φ be a reduction of a legal string u with $N = |\mathrm{snrdom}(\varphi)|$. Then $o(\mathcal{PC}_{\varphi(u)}) = o(\mathcal{PC}_u) - N$.

For a reduction φ of a legal string u, the difference between $\mathcal{R}_u$ and $\mathcal{R}_{\varphi(u)}$ is formulated in Theorem 6 in terms of reduction functions. We now reformulate this result for pointer-component graphs. The difference (up to isomorphism) between the pointer-component graph PC_1 of $\mathcal{R}_u$ and the pointer-component graph PC_2 of $rf_p(\mathcal{R}_u)$ (assuming rf_p is applicable to $\mathcal{R}_u$) is as follows: in PC_2 edge p is removed and also those vertices v that become isolated, except when v is the linear component (since the linear component always contains the source and target vertex). Since the only legal string u for which the linear component in $\mathcal{PC}_u$ is isolated is the empty string, in this case we obtain a graph containing only one vertex. This is formalized as follows. By abuse of notation we will also denote these functions as reduction functions rf_p.

Definition 13
For each edge p, we define the *p-reduction function* $rf_p : \mathsf{MGr} \to \mathsf{MGr}$, for $G = (V, E, \epsilon) \in \mathsf{MGr}$, by

$$rf_p(G) = (V', E', \epsilon|E'),$$

where $E' = E\backslash\{p\}$ and $V' = \{v \in V \mid v \in \epsilon(e) \text{ for some } e \in E'\}$ if $E' \neq \varnothing$, and $V' = \{\varnothing\}$ otherwise. ■

Therefore, these reduction functions correctly simulate (up to isomorphism) the effect of applications of a reduction functions on the underlying reduction graph when the reduction functions correspond to an applicable reduction. Note however, when these reduction functions do not correspond to an applicable reduction,

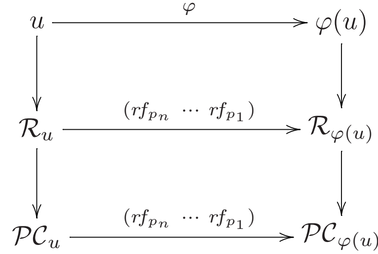

Figure 3.8: An illustration of Theorems 6 and 14 as a commutative diagram.

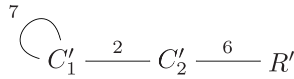

Figure 3.9: Pointer-component graph PC_1 from the Example.

the linear component may become isolated while there are still edges present. Thus in general the reduction functions for pointer-component graphs do not faithfully simulate the reduction functions for reduction graphs.

As a consequence of Theorem 6 we now obtain the following result.

Theorem 14
Let u be a legal string, and let φ be a reduction of u. Then

$$(rf_{p_n} \ \cdots \ rf_{p_2} \ rf_{p_1})(\mathcal{PC}_u) \approx \mathcal{PC}_{\varphi(u)},$$

where $\mathrm{dom}(\varphi) = \{p_1, p_2, \ldots, p_n\}$.

Thus, $\mathcal{PC}_{\varphi(u)}$ is obtained from $\mathcal{PC}_u$ (up to isomorphism) by iteratively removing the edges p_i *and* any isolated vertices that may appear after removing the edges. Thus the only difference between $\mathcal{PC}_{\varphi(u)}$ and $\mathcal{PC}_u|_D$ with $D = \mathrm{dom}(\varphi(u))$ is the possible existence of isolated vertices in $\mathcal{PC}_u|_D$. The only exception is the case $\varphi(u) = \lambda$, since we may not end up with the empty graph (without vertices), and thus one vertex should always remain. Figure 3.8 illustrates Theorems 6 and 14.

Example 11
We continue the example. We have $(\mathbf{snr}_4 \ \mathbf{sdr}_{5,3})(u) = 62\bar{7}726$. The pointer-component graph PC_1 of this legal string is shown in Figure 3.9. It is easy to see that the graph obtained by applying $(rf_5 \ rf_4 \ rf_3)$ to $\mathcal{PC}_u$ (Figure 3.7) is isomorphic to PC_1.

3.6 Spanning Trees in Pointer-Component Graphs

In this section we consider spanning trees in pointer-component graphs, and we show that there is an intimate connection between these trees and the Snr-orders of successful reductions. First we separate loops from other edges in pointer-component graphs.

Definition 15
Let u be a legal string and let $\mathcal{PC}_u = (V, E, \epsilon)$. We define $\text{bridge}(u) = \{e \in E \mid |\epsilon(e)| = 2\}$. ∎

Thus, bridge(u) is the set of vertex labels p for which there are vertices labelled by p in *different* connected components of $\mathcal{R}_u$.

Example 12
We continue the example. We have $\text{bridge}(u) = \{2, 3, 4, 5, 6\}$, and $\text{dom}(u) \setminus \text{bridge}(u) = \{7\}$. Indeed, the only loop in Figure 3.7 is 7, indicating that this pointer occurs only in one connected component of $\mathcal{R}_u$.

The following corollary to Theorem 14 observes that an edge in $\text{dom}(\varphi(u))$ is a loop in $\mathcal{PC}_{\varphi(u)}$ iff it is a loop in $\mathcal{PC}_u$.

Corollary 16
Let u be a legal string and φ a reduction of u. Then $\text{bridge}(\varphi(u)) = \text{dom}(\varphi(u)) \cap \text{bridge}(u) = \text{bridge}(u) \setminus \text{dom}(\varphi)$.

We now characterize $\{Spr, Sdr\}$-reductions in terms of pointer-component graphs.

Theorem 17
Let u be a legal string, and φ a reduction of u with $D = \text{dom}(\varphi(u))$. Then the following statements are equivalent:

1. φ is a $\{Spr, Sdr\}$-reduction,
2. $o(\mathcal{PC}_{\varphi(u)}) = o(\mathcal{PC}_u)$,
3. $\mathcal{PC}_{\varphi(u)} \approx \mathcal{PC}_u|_D$,
4. either $o(\mathcal{PC}_u|_D) = 1$ or $\mathcal{PC}_u|_D$ has no isolated vertices.

Proof
Statements (1) and (2) are equivalent by Theorem 12. If (3) holds, then clearly (2) holds. Assume now that (2) holds. Since an application of rf_p that does not remove vertices, only removes edge p, we have, by Theorem 14, $\mathcal{PC}_{\varphi(u)} \approx \mathcal{PC}_u|_D$. Thus (3) holds. Assume now that (3) holds. Since the pointer-component graph of a legal string u does not have isolated vertices except when $u = \lambda$, it follows that (4) holds. Finally, assume that (4) holds. If $o(\mathcal{PC}_u|_D) = 1$, then $o(\mathcal{PC}_u) = 1$ and therefore $1 \leq o(\mathcal{PC}_{\varphi(u)}) \leq o(\mathcal{PC}_u) = 1$. Consequently, $o(\mathcal{PC}_{\varphi(u)}) = o(\mathcal{PC}_u)$,

and (2) holds. On the other hand, if $\mathcal{PC}_u|_D$ has no isolated vertices, then by Theorem 14 the reduction functions corresponding to φ do not remove vertices, and hence (2) holds. ■

Thus, by Theorems 14 and 17, the effect of a rf_p operation where $p \in dom(\rho)$ for some applicable Spr or Sdr rule ρ is the removal of edge p. We now discuss the Snr case. By Lemma 7, we have the following result. Here v is identical to C in Lemma 7.

Lemma 18
Let u be a legal string and $p \in \mathrm{dom}(u)$. Then $\mathbf{snr}_p$ or $\mathbf{snr}_{\bar{p}}$ is applicable to u iff $p \in \mathrm{bridge}(u)$ and edge p in $\mathcal{PC}_u$ has an endpoint v such that (1) v is not the linear component and (2) p is the only edge with v as an endpoint (v is of degree 1).

Thus the effect of a rf_p operation where $p \in dom(\rho)$ for some applicable Snr rule ρ is the removal of edge p *and* the removal of vertex v as in Lemma 18. Vertex v is unique, otherwise $\mathcal{PC}_{\rho(u)}$ would have two vertices less than $\mathcal{PC}_u$ – a contradiction with Theorem 12.

The examples so far have shown connected pointer-component graphs. It turns out that these graphs are *always* connected.

Theorem 19
The pointer-component graph of any legal string is connected.

Proof
Let φ be a successful reduction of a legal string u (φ exists by Theorem 1). Assume that $\mathcal{PC}_u$ is not connected. Since $\mathcal{PC}_\lambda$ is connected, we have $\varphi = \varphi_2\rho\varphi_1$ for some reduction rule ρ, where $\mathcal{PC}_{\varphi_1(u)}$ is not connected, but $\mathcal{PC}_{\rho\varphi_1(u)}$ is. By the paragraph below Theorem 17, ρ cannot be a string double rule or a string positive rule, and therefore ρ is a string negative rule. By the paragraph below Lemma 18, $\mathcal{PC}_{\rho\varphi_1(u)}$ is obtained from $\mathcal{PC}_{\varphi_1(u)}$ by removing edge $p \in \mathrm{dom}(\rho)$ and removing one of the two endpoints of p. Therefore, $\mathcal{PC}_{\rho\varphi_1(u)}$ has the same number of connected components as $\mathcal{PC}_{\varphi_1(u)}$ – a contradiction. ■

The next theorem characterizes successfulness in $\{Snr\}$ using spanning trees.

Theorem 20
Let u be a legal string. Then u is successful in $\{Snr\}$ iff $\mathcal{PC}_u$ is a tree.

Proof
If u is successful in $\{Snr\}$, then, by Theorem 12, $\mathcal{PC}_u$ has $|\zeta|-1$ edges. Since $\mathcal{PC}_u$ has $|\zeta|$ vertices and is connected by Theorem 19, it follows that $\mathcal{PC}_u$ is a tree.

If $\mathcal{PC}_u$ is a tree, then $\mathcal{PC}_u$ has $|\zeta| - 1$ edges. Since the number of edges is $|\mathrm{dom}(u)|$, we have $|\mathrm{dom}(u)| = |\zeta| - 1$, and by Theorem 12 every $p \in \mathrm{dom}(u)$ is used in a string negative rule, and thus u is successful in $\{Snr\}$. ■

By Lemma 18 (and the paragraph below it), the possible orders of string negative rules applicable to the legal string u in Theorem 20 is restricted by the form of the tree $\mathcal{PC}_u$. Indeed, if we take the linear component of $\mathcal{R}_u$ as the root of $\mathcal{PC}_u$, then a successful reduction corresponds to a sequence of reduction functions that iteratively removes leaves and their connecting edges. We will discuss this in more detail in Section 3.9.

It turns out that the pointers on which string negative rules are applied in a successful reduction of a legal string u form a spanning tree of $\mathcal{PC}_u$.

Theorem 21
Let u be a legal string, and let $D \subseteq \mathrm{dom}(u)$. If there is a successful reduction φ of u with $\mathrm{snrdom}(\varphi) = D$, then $\mathcal{PC}_u|_D$ is a tree.

Proof
By postponing the string negative rules, there is a successful reduction $\varphi' = \varphi'_2\ \varphi'_1$ of u, where φ'_1 is a $\{Spr, Sdr\}$-reduction and φ'_2 is a $\{Snr\}$-reduction with $\mathrm{dom}(\varphi'_2) = D$. By Theorem 20, $\mathcal{PC}_{\varphi'_1(u)}$ is a tree. By Theorem 17 $\mathcal{PC}_{\varphi'_1(u)} \approx \mathcal{PC}_u|_D$. ∎

Example 13
We continue the example. We saw that $\varphi = \mathbf{snr}_6\ \mathbf{snr}_4\ \mathbf{snr}_2\ \mathbf{spr}_{\bar{7}}\ \mathbf{sdr}_{5,3}$ is a successful reduction of u. By Theorem 21, $\mathcal{PC}_u|_{\{2,4,6\}}$ is a tree. This is clear from Figure 3.7 where $\mathcal{PC}_u$ is depicted.

In the next few sections we prove the reverse implication of the previous theorem. This will require considerably more effort than the forward implication. The reason for this is that it is not obvious that when $\mathcal{PC}_u|_D$ is a tree, there is a reduction φ_1 of u such that $D = \mathrm{dom}(\varphi_1(u))$. We will use the pointer removal operation to prove this.

First, we consider a special case of the previous theorem. Since a loop can never be part of a tree, we have the following corollary to Theorem 21.

Corollary 22
Let u be a legal string and let $p \in \mathrm{dom}(u)$. If $p \in \mathrm{snrdom}(\varphi)$ for some (successful) reduction φ of u, then $p \in \mathrm{bridge}(u)$.

Example 14
We continue the example. Since $\varphi = \mathbf{snr}_6\ \mathbf{snr}_4\ \mathbf{snr}_2\ \mathbf{spr}_{\bar{7}}\ \mathbf{sdr}_{5,3}$ is a successful reduction of u, we have $2, 4, 6 \in \mathrm{bridge}(u)$.

We show in Theorem 30 below that the reverse implication of Corollary 22 also holds. Hence, the pointers $p \in \mathrm{bridge}(u)$ are exactly the pointers for which $\mathbf{snr}_p$ or $\mathbf{snr}_{\bar{p}}$ can occur in a (successful) reduction of u.

3.7 Merging and Splitting Components

In this section we consider the effect of pointer removal operations on pointer-component graphs. It turns out that these operations correspond to the merging

and splitting of connected components of the underlying reduction graph. We now introduce the merge operation on pointer-component graphs. Intuitively, the p-merge rule 'merges' the two endpoints of edge p into one vertex, and therefore the resulting graph has exactly one vertex less than the original graph. Formally the merge operation is as follows.

Definition 23
For each edge p, the *p-merge rule*, denoted by merge_p, is a rule applicable to (i.e., defined on) $G = (V, E, \epsilon) \in \mathsf{MGr}$ with $p \in E$ and $|\epsilon(p)| = 2$. It is defined by

$$\text{merge}_p(G) = (V', E', \epsilon'),$$

where $E' = E \backslash \{p\}$, $V' = (V \backslash \epsilon(p)) \cup \{v'\}$ with a new vertex $v' \notin V$, and $\epsilon'(e) = \{h(v_1), h(v_2)\}$ iff $\epsilon(e) = \{v_1, v_2\}$ where $h(v) = v'$ if $v \in \epsilon(p)$, otherwise it is the identity. ∎

Again, we allow $v_1 = v_2$ in the previous definition. Note that p-merge rules commute under composition. Thus, if $(\text{merge}_q\ \text{merge}_p)$ is applicable to G, then

$$(\text{merge}_q\ \text{merge}_p)(G) = (\text{merge}_p\ \text{merge}_q)(G).$$

Theorem 24
Let $G = (V, E, \epsilon) \in \mathsf{MGr}$, and let $D = \{p_1, \dots, p_n\} \subseteq E$. Then $(\text{merge}_{p_n} \cdots \text{merge}_{p_1})$ is applicable to G iff $G|_D$ is acyclic.

Proof
$(\text{merge}_{p_n} \cdots \text{merge}_{p_1})$ is applicable on G iff for all p_i $(1 \le i \le n)$, $\epsilon(p_i) \not\subseteq \epsilon(\{p_1, \dots, p_{i-1}\})$ and $|\epsilon(p_i)| = 2$. Furthermore, the latter holds iff $G|_D$ is acyclic. ∎

Surprisingly, the pointer removal operation is crucial in the proofs of the main results. The next theorem compares $\mathcal{PC}_u$ with $\mathcal{PC}_{\text{rem}_{\{p\}}(u)}$ for a legal string u and $p \in \text{dom}(u)$. We distinguish three cases: either the number of vertices of $\mathcal{PC}_{\text{rem}_{\{p\}}(u)}$ is one less, is equal, or is one more than the number of vertices of $\mathcal{PC}_u$. The proof of this theorem shows that the first case corresponds to merging two connected components of $\mathcal{R}_u$ into one connected component, and the last case corresponds to splitting one connected component of $\mathcal{R}_u$ into two connected components.

Theorem 25
Let u be a legal string.

- If $p \in \text{bridge}(u)$, then $\mathcal{PC}_{\text{rem}_{\{p\}}(u)} \approx \text{merge}_p(\mathcal{PC}_u)$ (and therefore $o(\mathcal{PC}_{\text{rem}_{\{p\}}(u)}) = o(\mathcal{PC}_u) - 1$).
- If $p \in \text{dom}(u) \backslash \text{bridge}(u)$, then $o(\mathcal{PC}_u) \le o(\mathcal{PC}_{\text{rem}_{\{p\}}(u)}) \le o(\mathcal{PC}_u) + 1$.

Proof
Consider $p \in \text{bridge}(u)$ first. Then the two desire edges with vertices labelled by p belong to different connected components of $\mathcal{R}_u$. We distinguish two cases:

whether or not there are cyclic components consisting of only vertices labelled by p.

If there is cyclic component consisting of only vertices labelled by p, then by Lemma 7, pp or $\bar{p}\bar{p}$ are substrings of u, and $\mathcal{R}_u$ is

$$p \;=\!=\!=\; p$$

$$\cdots \text{———} q_1 =\!=\!= p \text{———} p =\!=\!= q_2 \text{———} \cdots$$

where we omitted the parts of the graph that are the same compared to $\mathcal{R}_{rem_{\{p\}}(u)}$. Now, $\mathcal{R}_{rem_{\{p\}}(u)}$ is

$$\cdots \text{———} q_1 =\!=\!= q_2 \text{———} \cdots$$

Therefore $\mathcal{PC}_{\mathrm{rem}_{\{p\}}(u)}$ can be obtained (up to isomorphism) from $\mathcal{PC}_u$ by applying the merge_p operation.

Now assume that there are no cyclic components consisting of only vertices labelled by p. Then, $\mathcal{R}_u$ is

$$\cdots \text{———} q_1 =\!=\!= p \text{———} p =\!=\!= q_2 \text{———} \cdots$$

$$\cdots \text{———} q_3 =\!=\!= p \text{———} p =\!=\!= q_4 \text{———} \cdots$$

where we again omitted the parts of the graph that are the same compared to $\mathcal{R}_{rem_{\{p\}}(u)}$. Now, depending on the positions of $q_1, \ldots, q_4$ relative to p in u and on whether p is positive or negative in u, $\mathcal{R}_{rem_{\{p\}}(u)}$ is either

$$\cdots \text{———} q_1 =\!=\!= q_4 \text{———} \cdots$$

$$\cdots \text{———} q_3 =\!=\!= q_2 \text{———} \cdots$$

or

$$\cdots \text{———} q_1 =\!=\!= q_3 \text{———} \cdots$$

$$\cdots \text{———} q_4 =\!=\!= q_2 \text{———} \cdots$$

Note that since a desire edge connects two ‘segments’ (each represented by a reality edge) corresponding to different occurrences of p or $\bar{p}$, it is not possible that q_1 and q_2 are connected by a reality edge (and also for q_3 and q_4) in $\mathcal{R}_{rem_{\{p\}}(u)}$. Therefore, we have only the above two cases. Since q_1 and q_2 remain part of the same connected component (the same holds for q_3 and q_4), in both cases the two connected components are merged, and thus $\mathcal{PC}_{\mathrm{rem}_{\{p\}}(u)}$ can be obtained (up to isomorphism) from $\mathcal{PC}_u$ by applying the merge_p operation.

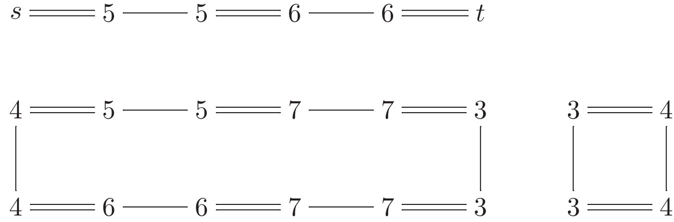

Figure 3.10: Reduction graph $\mathcal{R}_{rem_{\{2\}}(u)}$ from the Example.

Finally, consider $p \in \text{dom}(u) \backslash \text{bridge}(u)$. Then the two desire edges with vertices labelled by p belong to the same connected component of $\mathcal{R}_u$. By Lemma 7, there are no cyclic components consisting of four vertices which are all labelled by p. We can distinguish two cases: whether or not there is a reality edge e connecting two vertices labelled by p. If there is such a reality edge e then $\mathcal{R}_u$ is

$$\cdots \text{———} q_1 = p \text{———} p = p \text{———} p = q_4 \text{———} \cdots$$

This occurs precisely when $\bar{p}p$ or $p\bar{p}$ is a substring of u. Now, $\mathcal{R}_{rem_{\{p\}}(u)}$ is

$$\cdots \text{———} q_1 = q_4 \text{———} \cdots$$

Therefore, $\mathcal{R}_{rem_{\{p\}}(u)}$ has $N = o(\mathcal{PC}_u)$ cyclic components.

If there is no such a reality edge e, then $\mathcal{R}_u$ is

$$\cdots \text{———} q_1 = p \text{———} p = q_2 \text{———} L \text{———} q_3 = p \text{———} p = q_4 \text{———} \cdots$$

where L represents some (possibly empty) 'linear subgraph' of $\mathcal{R}_u$. Now, $\mathcal{R}_{rem_{\{p\}}(u)}$ is either

$$\cdots \text{———} q_4 = q_2 \text{———} L \text{———} q_3 = q_1 \text{———} \cdots$$

or

$$L, \quad L \text{———} q_2, \quad L \text{———} q_3, \quad q_2 = q_3 \qquad \cdots \text{———} q_1 = q_4 \text{———} \cdots$$

Therefore, $\mathcal{R}_{rem_{\{p\}}(u)}$ has either N cyclic components (corresponding with the first case) or $N + 1$ cyclic components (corresponding with the second case). ■

Example 15

We continue the example. By Theorem 25, we know from Figure 3.7 that $\mathcal{PC}_{\text{rem}_{\{2\}}(u)} \approx \text{merge}_2(\mathcal{PC}_u)$, merging components C_1 and C_2. Indeed, this is transparent from Figures 3.7, 3.10 and 3.11, where $\mathcal{PC}_u$, $\mathcal{R}_{rem_{\{2\}}(u)}$, and $\mathcal{PC}_{\text{rem}_{\{2\}}(u)}$ are depicted, respectively.

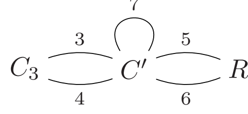

Figure 3.11: $\mathcal{PC}_{\mathrm{rem}_{\{2\}}(u)}$ from the Example.

Again by Theorem 25, we know from Figure 3.10 that $\mathcal{R}_{rem_{\{2,7\}}(u)}$ has two or three cyclic components. Indeed, this is transparent from Figure 3.5, where $\mathcal{R}_{rem_{\{2,7\}}(u)}$ is depicted.

Note that by the definition of merge_p, merge_p is applicable to $\mathcal{PC}_u$ precisely when $p \in \mathrm{bridge}(u)$. Therefore, by Theorems 24 and 25, we have the following corollary.

Corollary 26
Let u be a legal string, and let $D \subseteq \mathrm{dom}(u)$. If $\mathcal{PC}_u|_D$ is acyclic, then

$$\mathcal{PC}_{\mathrm{rem}_D(u)} \approx (\mathrm{merge}_{p_n} \ \cdots \ \mathrm{merge}_{p_1})(\mathcal{PC}_u),$$

where $D = \{p_1, \ldots, p_n\}$.

3.8 Applicability of the String Negative Rule

In this section we characterize for a given set of pointers D, whether or not there is a (successful) strategy that applies string negative rules on exactly these pointers. First we will prove the following result which depends heavily on the results of the previous section. The forward implication of the result states that by removing pointers from u that form a spanning tree in $\mathcal{PC}_u$ we obtain a legal string u' for which the reduction graph does not have cyclic components.

Lemma 27
Let u be a legal string, and let $D \subseteq \mathrm{dom}(u)$. Then $\mathcal{PC}_u|_D$ is a tree iff $\mathcal{R}_{rem_D(u)}$ and $\mathcal{R}_u$ have 0 and $|D|$ cyclic components, respectively.

Proof
We first prove the forward implication. Let $\mathcal{PC}_u|_D$ be a tree. By Corollary 26, $\mathcal{PC}_{\mathrm{rem}_D(u)}$ contains a single vertex. Thus $\mathcal{R}_{rem_D(u)}$ has no cyclic components. Since $\mathcal{PC}_u|_D$ is a tree, we have $|D| = |\zeta| - 1$.

We now prove the reverse implication. Let $\mathcal{R}_{rem_D(u)}$ not contain cyclic components and $|D| = |\zeta| - 1$. By Theorem 25 we see that the removal of each pointer p in D corresponds to a merge_p operation, otherwise $\mathcal{R}_{rem_D(u)}$ would contain cyclic components. Therefore, $(\mathrm{merge}_{p_n} \cdots \mathrm{merge}_{p_1})$ is applicable to $\mathcal{PC}_u$ with $D = \{p_1, \ldots, p_n\}$. Therefore, by Theorem 24, $\mathcal{PC}_u|_D$ is acyclic. Again since $|D| = |\zeta| - 1$, it is a tree. ■

$$C_3 \overset{3}{\text{———}} C_1 \overset{2}{\text{———}} C_2 \overset{5}{\text{———}} R$$

Figure 3.12: A subgraph of the pointer-component graph from the Example.

$$\begin{array}{ccccc} C_1 & & & & \\ 3 \Big| & \diagdown^{2} & & & \\ C_3 & \underset{4}{\text{———}} & C_2 & \qquad & R \end{array}$$

Figure 3.13: A subgraph of the pointer-component graph from the Example.

$$\begin{array}{ccccccccccccccc} s & = & 4 & — & 4 & = & 6 & — & 6 & = & 7 & — & 7 \\ & & & & & & & & & & & & \| \\ t & = & 6 & — & 6 & = & 7 & — & 7 & = & 4 & — & 4 \end{array}$$

Figure 3.14: The reduction graph $\mathcal{R}_{rem_{D_1}(u)}$ from the Example.

$$s = 5 — 5 = 6 — 6 = t$$

$$\begin{array}{ccccccc} 5 & = & 7 & — & 7 & = & 6 \\ | & & & & & & | \\ 5 & = & 7 & — & 7 & = & 6 \end{array}$$

Figure 3.15: The reduction graph $\mathcal{R}_{rem_{D_2}(u)}$ from the Example.

Example 16
We continue the previous example. Let $D_1 = \{2, 3, 5\}$ and $D_2 = \{2, 3, 4\}$. Then $\mathcal{PC}_u|_{D_1}$ ($\mathcal{PC}_u|_{D_2}$, resp.) is given in Figure 3.12 (Figure 3.13, resp.). Notice that $|D_1| = |D_2| = |\zeta| - 1$. Since $\mathcal{PC}_u|_{D_1}$ is a tree and $\mathcal{PC}_u|_{D_2}$ is not a tree, by Lemma 27, it follows that $\mathcal{R}_{rem_{D_1}(u)}$ does not have cyclic components and that $\mathcal{R}_{rem_{D_2}(u)}$ does have at least one cyclic component. This is illustrated in Figures 3.14 and 3.15, where $\mathcal{R}_{rem_{D_1}(u)}$ and $\mathcal{R}_{rem_{D_2}(u)}$ are depicted respectively.

The next theorem is one of the main results of this chapter. It follows directly from Lemma 27 and Lemma 10, and improves Theorem 21 by characterizing exactly which string negative rules can be applied together in a successful reduction of a given legal string.

Theorem 28
Let u be a legal string, and let $D \subseteq \mathrm{dom}(u)$. There is a successful reduction φ of u with $\mathrm{snrdom}(\varphi) = D$ iff $\mathcal{PC}_u|_D$ is a tree.

Since there are many well known and efficient methods for determining spanning trees in a graph, it is easy to determine, for a given set of pointers D, whether or not there is a successful reduction applying string negative rules on exactly the pointers of D (for a given legal string u).

Example 17
We continue the example. By Theorem 28 and Figure 3.12, there is a successful reduction φ of u with $\mathrm{snrdom}(\varphi) = \{2, 3, 5\}$. Indeed, we can take for example $\varphi = \mathbf{snr}_5\ \mathbf{snr}_2\ \mathbf{snr}_{\bar{3}}\ \mathbf{spr}_{\bar{7}}\ \mathbf{sdr}_{4,6}$.

By Theorem 28 (or Theorem 21) and Figure 3.13, there is no successful reduction φ of u with $\mathrm{snrdom}(\varphi) = \{2, 3, 4\}$. For example, $(\mathbf{spr}_5\ \mathbf{spr}_7)(u) = 62\bar{3}\bar{4}\bar{2}346$ and thus there is no string pointer rule for pointer 6 applicable to this legal string.

In the next corollary we consider the more general case $|D| \leq |\zeta| - 1$, instead of $|D| = |\zeta| - 1$ in Theorem 28, i.e., we consider acyclic graphs rather than trees.

Corollary 29
Let u be a legal string, and let $D \subseteq \mathrm{dom}(u)$. There is a (successful) reduction φ of u with $D \subseteq \mathrm{snrdom}(\varphi)$ iff $\mathcal{PC}_u|_D$ is acyclic.

Proof
We first prove the forward implication. By Theorem 28, $\mathcal{PC}_u|_D$ is a subgraph of a tree, and therefore acyclic.

We now prove the reverse implication. By Theorem 19, $\mathcal{PC}_u$ is connected, and since $\mathcal{PC}_u|_D$ does not contain cycles, we can add edges $q \in \mathrm{dom}(u) \backslash D$ from $\mathcal{PC}_u$ such that the resulting graph is a tree. Then by Theorem 28, it follows that there is a (successful) reduction φ of u with $D \subseteq \mathrm{snrdom}(\varphi)$. ■

The previous corollary with $|D| = 1$ shows that the reverse implication of Corollary 22 also holds, since $\mathcal{PC}_u|_{\{p\}}$ acyclic implies that the edge p connects two different vertices in $\mathcal{PC}_u$.

Theorem 30
Let u be a legal string and let $p \in \mathrm{dom}(u)$. Then $p \in \mathrm{snrdom}(\varphi)$ for some (successful) reduction φ of u iff $p \in \mathrm{bridge}(u)$.

This theorem can also be proven directly.

Proof
To prove the reverse implication, let no reduction of u contain either $\mathbf{snr}_p$ or $\mathbf{snr}_{\bar{p}}$. We prove that $p \notin \mathrm{bridge}(u)$. By iteratively applying **snr**, **spr** and **sdr** on pointers that are not equal to p or $\bar{p}$, we can reduce u to a legal string v such that for all $q \in \mathrm{dom}(v)\backslash\{p\}$:

- qq and $\bar{q}\bar{q}$ are not substrings of v.
- q is negative in v.
- q does not overlap with any pointer in $\mathrm{dom}(v)\backslash\{p\}$.

If $\mathrm{rem}_{\{p\}}(v) = \lambda$, then v is equal to either $p\bar{p}$, $\bar{p}p$, pp or $\bar{p}\bar{p}$. If $\mathrm{rem}_{\{p\}}(v) \neq \lambda$, then, by the last two conditions, there is a $q \in \Pi$ such that qq is a substring of $\mathrm{rem}_{\{p\}}(v)$. Then, by the first condition, either qpq, $q\bar{p}q$, $qp\bar{p}q$, $q\bar{p}pq$, $qppq$ or $q\bar{p}\bar{p}q$ is a substring of v.

Thus, either qpq, $q\bar{p}q$, $p\bar{p}$, $\bar{p}p$, pp or $\bar{p}\bar{p}$ is a substring of v. Since no reduction of u contains $\mathbf{snr}_p$ or $\mathbf{snr}_{\bar{p}}$, the last two cases are not possible. The first two cases correspond to the following part of $\mathcal{R}_v$.

$$\cdots = p - p = q - q = p - p = \cdots$$

The cases where $p\bar{p}$ or $\bar{p}p$ is a substring of v correspond to the following part of $\mathcal{R}_v$

$$\cdots = p - p = p - p = \cdots$$

Consequently, in either case, the two desire edges of $\mathcal{R}_v$ with vertices labelled by p belong to the same connected component. Thus $p \notin \mathrm{bridge}(v)$. By Corollary 16, $p \notin \mathrm{bridge}(u)$. ■

3.9 The Order of Loop Recombination

According to Theorem 28 a set D of pointers can occur as the domain of Snr rules in a successful reduction of a legal string u exactly when the graph $\mathcal{PC}_u|_D$ is a tree. This result can be strengthened to incorporate the order in which the Snr rules are applied. We show that in a successful reduction φ we can only apply Snr rules in orderings determined by the tree $\mathcal{PC}_u|_D$ with the linear component as root, where D is the domain of Snr rules in φ. These orderings are similar topological orderings in a directed acyclic graph, however, here we order the edges instead of the vertices.

Definition 31
Let $T = (V, E, \epsilon)$ be a tree, and let $R \in V$. We define the relation $\prec_{T,R}$ over E as follows. For $e_1, e_2 \in E$, we have $e_1 \prec e_2$ iff $\epsilon(e_1) = \{C_x, C_y\}$, $\epsilon(e_2) = \{C_y, C_z\}$, and C_y (C_z, resp.) is the father of C_x (C_y, resp.) in T considering R as the root of T. Also, an *edge-topological ordering of T (with root R)* is a topological ordering of $\prec_{T,R}$. ∎

Example 18
We continue the example. Consider again tree $\mathcal{PC}_u|_{D_1}$ shown in Figure 3.12. Taking R as the root of $\mathcal{PC}_u|_{D_1}$, it follows that $(3, 2, 5)$ is an edge-topological ordering of $\mathcal{PC}_u|_{D_1}$.

The next theorem characterizes exactly the possible orderings in which string negative rules can be applied in a successful reduction of a given legal string.

Theorem 32
Let u be a legal string, let L be a linear ordering of a subset L' of dom(u). There is a successful reduction φ of u with *Snr*-order L iff $\mathcal{PC}_u|_{L'}$ is a tree, where L is an edge-topological ordering of $\mathcal{PC}_u|_{L'}$ with the linear component R of $\mathcal{R}_u$ as root.

Proof
Let $L = (p_1, p_2, \ldots, p_n)$. We first prove the forward implication. Recall that we can postpone the application of string negative rules, thus $\mathbf{snr}_{\tilde{p}_n}\ \mathbf{snr}_{\tilde{p}_{n-1}} \cdots \mathbf{snr}_{\tilde{p}_1}\ \varphi'$ is also a successful reduction of u, where φ' is a $\{Spr, Sdr\}$-reduction and $\tilde{p}_i \in \{p_i, \bar{p}_i\}$ for $i \in \{1, \ldots, n\}$. By Theorem 28, $\mathcal{PC}_u|_{L'}$ is a tree.

We prove that L is an edge-topological ordering of $\mathcal{PC}_u|_{L'}$ with root R. By Theorem 17, $\mathcal{PC}_{\varphi'(u)} \approx \mathcal{PC}_u|_{L'}$. If $n > 0$, then $\mathbf{snr}_{\tilde{p}_1}$ is applicable to $\varphi'(u)$. By Lemma 18, edge p_1 is connected to a leaf of $\mathcal{PC}_{\varphi'(u)}$. By Theorem 14 and the paragraph below Lemma 18, $\mathcal{PC}_{(\mathbf{snr}_{\tilde{p}_1}\ \varphi')(u)}$ is isomorphic to the graph obtained from $\mathcal{PC}_{\varphi'(u)}$ by removing p_1 and its leaf. Now (assuming $n > 1$), since $\mathbf{snr}_{\tilde{p}_2}$ is applicable to $(\mathbf{snr}_{\tilde{p}_1}\ \varphi')(u)$, p_2 is connected to a leaf in $\mathcal{PC}_{(\mathbf{snr}_{\tilde{p}_1}\ \varphi')(u)}$. By iterating this argument, it follows that L is an edge-topological ordering of $\mathcal{PC}_{\varphi'(u)} \approx \mathcal{PC}_u|_{L'}$ with root R.

We now prove the reverse implication. Since $\mathcal{PC}_u|_{L'}$ is a tree, by Theorem 28 there is a successful reduction $\varphi = \varphi_2\ \varphi_1$ of u, where φ_1 is a $\{Spr, Sdr\}$-reduction and φ_2 is a $\{Snr\}$-reduction with dom(φ_2) $= L'$. Let L be an edge-topological ordering of $\mathcal{PC}_u|_{L'}$ with the linear component R of $\mathcal{R}_u$ as root. Again, by Theorem 17, $\mathcal{PC}_u|_{L'} \approx \mathcal{PC}_{\varphi_1(u)}$.

If $n > 0$, then p_1 is connected to a leaf of $\mathcal{PC}_{\varphi_1(u)}$. By Lemma 18, $\mathbf{snr}_{\tilde{p}_1}$ is applicable to $\varphi_1(u)$ for some $\tilde{p_1} \in \{p_1, \bar{p_1}\}$. Again by Theorem 14 and the paragraph below Lemma 18, $\mathcal{PC}_{(\mathbf{snr}_{\tilde{p}_1}\ \varphi_1)(u)}$ is isomorphic to the graph obtained from $\mathcal{PC}_{\varphi_1(u)}$ by removing p_1 and its leaf. By iterating this argument, it follows that $\mathbf{snr}_{\tilde{p}_n}\ \mathbf{snr}_{\tilde{p}_{n-1}} \cdots \mathbf{snr}_{\tilde{p}_1}$ is a successful reduction of u for some $\tilde{p_i} \in \{p_i, \bar{p_i}\}$ and $1 \le i \le n$ with $n \ge 0$. ∎

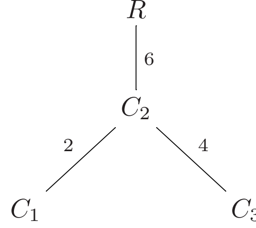

Figure 3.16: A subgraph of the pointer-component graph from the Example.

Example 19
We continue the example. Since $(3,2,5)$ is an edge-topological ordering of tree $\mathcal{PC}_u|_{D_1}$ with root R, by Theorem 32, there is a successful reduction φ of u with *Snr*-order $(3,2,5)$. Indeed, we can take for example $\varphi = \mathbf{snr}_5\ \mathbf{snr}_2\ \mathbf{snr}_{\bar{3}}\ \mathbf{spr}_{\bar{7}}\ \mathbf{sdr}_{4,6}$.

We say that two reduction rules ρ_1 and ρ_2 can be applied in *parallel* to u if both $\rho_2\ \rho_1$ and $\rho_1\ \rho_2$ are applicable to u (see [18]).

Corollary 33
Let u be a legal string, and $p, q \subseteq \mathrm{dom}(u)$ with $p \neq q$. Then $\mathbf{snr}_{\tilde{p}}$ and $\mathbf{snr}_{\tilde{q}}$ can be applied in parallel to u for some $\tilde{p} \in \{p, \bar{p}\}$, $\tilde{q} \in \{q, \bar{q}\}$ iff there is a spanning tree T in $\mathcal{PC}_u$ such that p and q both connect to leaves (considering the linear component of $\mathcal{R}_u$ as the root).

Let R be the linear component of $\mathcal{R}_u$. Clearly, for spanning tree T in $\mathcal{PC}_u$ with root R that contains edges p and q, we have the following: p and q in T are independent for $\prec_{T,R}$ iff there is no simple path in T from R to another vertex of T containing both edges p and q. The next corollary considers the case whether or not $\mathbf{snr}_{\tilde{p}}$ and $\mathbf{snr}_{\tilde{q}}$ can *eventually* be applied in parallel.

Corollary 34
Let u be a legal string, and $p, q \subseteq \mathrm{dom}(u)$ with $p \neq q$. Then $\mathbf{snr}_{\tilde{p}}$ and $\mathbf{snr}_{\tilde{q}}$ can be applied in parallel to $\varphi(u)$ for some $\tilde{p} \in \{p, \bar{p}\}$, $\tilde{q} \in \{q, \bar{q}\}$, and some reduction φ of u iff there is a spanning tree T in $\mathcal{PC}_u$ containing both edges p and q, where p and q are independent for $\prec_{T,R}$.

Example 20
We continue the example. Let $D_3 = \{2,4,6\}$. Then in the tree $\mathcal{PC}_u|_{D_3}$, depicted in Figure 3.16, there is no simple path from R to another vertex of $\mathcal{PC}_u|_{D_3}$ containing both edges 2 and 4. By Corollary 34, $\mathbf{snr}_{\tilde{2}}$ and $\mathbf{snr}_{\tilde{4}}$ can be applied in parallel to $\varphi(u)$ for some $\tilde{2} \in \{2, \bar{2}\}$, $\tilde{4} \in \{4, \bar{4}\}$, and some reduction φ of u. Indeed, if we take $\varphi = \mathbf{spr}_{\bar{7}}\ \mathbf{sdr}_{3,5}$, then $\mathbf{snr}_2$ and $\mathbf{snr}_4$ can be applied in parallel to $\varphi(u) = 622446$.

3.10 Conclusion

This chapter shows that one can efficiently determine the possible sequences of loop recombination operations that can be applied in the transformation of a given gene from its micronuclear to its macronuclear form. Formally, one can determine the orderings of string negative rules that can be present in successful reductions of u. This is a characterization in terms of a graph defined on the (components of the) reduction graph. Future research could focus on similar characterizations for the string positive rules and the string double rules. However, this would require other concepts, since the pointer-component graph does not retain information regarding positiveness or overlap of pointers, notions crucial for the applicability of the other two operations.